제36회 한국정보처리학회 추계학술발표대회 논문집 제18권 제2호 (2011. 11)

가상 환경에서 시멘틱 갭 연결을 통한 자원 중복성 제거

김인혁, 김태형, 엄영익
성균관대학교 전자전기컴퓨터공학과
e-mail:{kkojiband, kim15m, yieom}@ece.skku.ac.kr

Resource Redundancy Elimination by Bridging the Semantic Gap in Virtualized Systems

Inhyeok Kim, Taehyoung Kim, Young Ik Eom
School of Info. and Comm. Eng., Sungkyunkwan University

요 약

전통적인 가상화 기술들은 하나의 물리 머신에서 서로 다른 운영체제들을 동작시키기 위해 여러 개의 가상 머신을 제공하고 있다. 이러한 가상화 기술들은 운영체제를 소스 수정없이, 혹은 약간의 수정만으로 물리 머신에서 동작하는 것과 동일한 방식으로 동작할 수 있도록 지원하기 때문에 호스트와 게스트 간의 깊은 시멘틱 갭을 발생시킨다. 이러한 시멘틱 갭은 여러 컴퓨팅 자원에 대한 공간 중복과 접근 중복의 중요한 원인이 되고, 이러한 자원 중복은 대규모 가상화 시스템의 확장성에 제약한다. 이러한 자원 중복을 제거하기 위해 컨텐츠 기반 페이지 공유 등 다양한 연구들이 진행되어 왔지만 운영체제 수정없이 지원하는 정책으로 인한 시멘틱 갭은 여전히 제안 기법들을 제한하게 된다. 이에 우리들은 운영체제의 자원 관리 부분을 수정하여 근본적인 시멘틱 갭을 제거함으로써 메모리와 스토리지의 공간 및 접근 중복성을 제거할 수 있는 파일시스템을 제안하였다. 그리고 실험을 통해 기존의 가상 블록 장치를 사용하는 파일시스템과 비교 및 평가하여 제안 시스템이 페이지 캐시 공유 및 스토리지 접근 중복 제거에 효과적임을 입증하였다.

1. 서론

1960년대부터 메인프레임 서버의 활용률을 높이기 위해 에뮬레이션, 반/전가상화와 같은 가상화 기술들이 꾸준히 연구되어 왔다. 하지만 이러한 접근 방식은 가상화 환경을 제공하기 위해 호스트와 게스트 간의 소프트웨어 계층을 완전히 분리시킴으로써 깊은 시멘틱 갭을 유발시키고, 그로 인해 여러 종류의 자원 중복성 문제가 발생한다[1]. 특히, 메모리와 스토리지의 자원 중복성 문제는 대규모 가상화 시스템의 확장성에 큰 제약 사항으로 작용한다. 메모리의 자원 중복성을 제거하기 위해 컨텐츠 기반 페이지 공유와 공유 기반 블록 장치 등의 관련 연구들이 소개되었지만, 시멘틱 갭으로 인한 전체 메모리 스캔 등 실행시 큰 오버헤드로 인해 사용에 한계가 있다. 그래서 우리는 시멘틱 갭 연결을 통해 메모리와 스토리지의 자원 중복성을 제거하는 호스트 기반 파일시스템을 제안하였고, 실험을 통해 제안시스템이 페이지 공유 및 가상 블록 장치 우회에 효과적임을 보여주었다[2, 3].

본 논문의 구성은 다음과 같다. 2장에서는 자원 중복에 대한 정의와 설명을 보여주고 3장에서는 관련 연구를 소개한다. 4장에서는 제안 기법인 호스트 기반 파일시스템에 대해 소개하고, 5장에서는 실험 결과를 보여주고 있다. 마지막으로 6장에서는 결론을 포함하고 있다.

이 논문은 2011년도 정부(교육과학기술부)의 재원으로 한국연구재단-차세대정보컴퓨팅기술개발사업의 지원을 받아 수행된 연구임 (No. 2011-0020520)

2. 자원 중복

컴퓨터 자원은 크게 시간 공유 자원 및 공간 공유 자원, 두 가지로 분류할 수 있다. 시간 공유 자원은 모든 사용자들이 하나의 자원을 배타적으로 사용하는 경우를 말한다. 전통적인 시간 공유 자원은 프로세서와 네트워크 장치와 같이 해당 자원을 사용하는 모든 프로세스들이 주어진 스케줄링 정책에 의해 순서대로 자원을 선점하여 사용한다. 다음과 같이 시간 공유 자원 r의 총 자원 사용량 $T(r)$은 i번째 프로세스의 자원 사용량인 $U(r)_i$을 이용하여 계산할 수 있다.

$$T(r) = \sum_{i=1}^{n} U(r)_i$$

이와 같이 시간 공유 자원의 총 사용량은 간단히 모든 프로세스의 자원 사용량의 합으로 계산할 수 있다. 반면에 공간 공유 자원의 총 자원 사용량은 정확한 계산이 불가능하다. 또한, 메모리와 같은 공간 공유 자원은 시스템 성능 향상을 위해 가능한 많은 양의 캐시를 저장하려고 하고, 아직 할당되지 않은 영역들도 존재한다. 그렇기 때문에 공간 공유 자원의 총 자원 사용량 $T(r)$은 i번째 프로세스의 자원 사용량인 $U(r)_i$, 캐시 사용량인 $C(r)_i$, 미할당 자원량인 $F(r)_i$를 이용하여 다음과 같이 나타낼 수 있다.

$$T(r) = \sum_{i=1}^{n} U(r)_i + \sum_{i=1}^{n} C(r)_i + \sum_{i=1}^{n} F(r)_i$$

가까운 미래에 사용될 가능성이 있는 캐시의 크기를 알아내는 것은 해당 자원을 직접적으로 관리하는 운영체제에서도 어려운 일이다. 게다가 가상 환경에서 게스트들을 관리하는 호스트에게는 더욱 어려운 문제가 된다. 일반 환경에서 운영체제는 LRU와 같은 정책을 이용하여 캐시를 교체하는 기법을 사용하지만, 가상 환경에서 호스트는 시멘틱 갭에 의해 기본적인 LRU와 같은 정책조차도 사용할 수 없다. 이러한 문제는 운영체제가 일반 환경에서 자원을 독점적으로 사용할 때는 큰 문제가 되지 않지만 가상 환경에서 하나의 자원을 여러 가상 머신들과 분배해야 되는 상황에서는 심각한 문제를 유발하게 된다. 즉, 어떤 가상 머신의 불필요한 캐시와 미할당 공간들이 다른 가상 머신을 위해 유용하게 사용될 수도 있기 때문이다. 이러한 제약사항은 수백, 수천개의 가상 머신을 동시에 운영하는 시스템에서는 확장성에 큰 문제를 초래할 수 있다.

3. 관련 연구

VMware ESX server[4]는 공유 가능한 페이지를 찾기 위해 주기적으로 게스트의 전체 메모리를 스캔하는 기법을 사용하고 있다. 잦은 전체 메모리 스캔은 실행시 시스템에 큰 부하를 주기 때문에 자주 수행하지 못 하고, 이로 인해 짧은 주기를 가지는 페이지들은 공유될 수 있는 기회를 가지지 못하는 문제가 발생한다.

Satori[5]는 가상 환경에서 효과적인 페이지 공유를 위하여 공유 기반 블록 장치를 소개하고 있다. 이는 모든 게스트들이 하나의 파일시스템 이미지를 공유하여 특정 페이지 캐시가 사용하는 블록의 위치로 공유 여부를 판단한다. 이를 통해 Satori는 적은 비용으로 짧은 주기의 페이지 캐시도 공유가 가능하지만, 모든 게스트들이 동일한 파일시스템 이미지를 사용해야 한다는 문제가 있고, 게스트가 임의의 블록의 내용을 변경하게 되면 copy-on-write 기법에 의해 해당 블록은 앞으로 블록 위치를 이용하여 간단하게 공유될 수 있는 기회는 잃어버리고, VMware와 같이 컨텐츠 기반 페이지 공유 기법에 의해서만 공유될 수 있게 된다.

4. 호스트 기반 파일시스템

우리는 호스트와 게스트의 시멘틱 갭을 제거함으로써 효과적으로 페이지 캐시와 스토리지를 공유 및 관리하는 새로운 파일시스템을 제안하였다. 기존의 파일시스템은 가상 블록 장치를 사용하기 때문에 호스트는 외부에서 게스트의 파일시스템의 논리적인 구조를 알 수 없다. 하지만 제안 파일시스템은 가상 블록 장치를 사용하지 않고 호스트의 파일시스템 일부분을 게스트가 직접 사용하기 때문에 호스트와 게스트는 파일시스템의 논리적인 구조를 똑같이 볼 수 있게 된다. 호스트 기반 파일시스템의 구조는 다음과 같다.

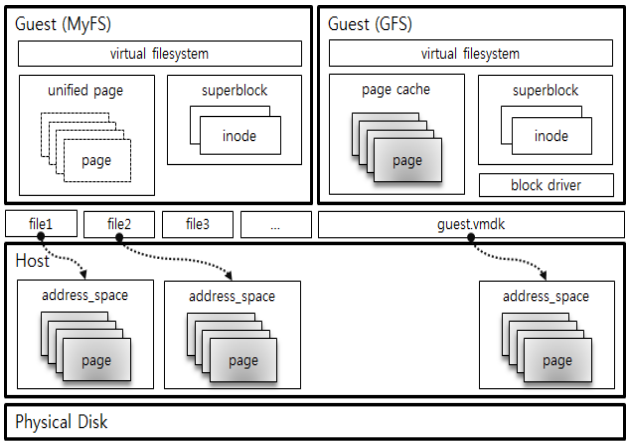

그림 1 호스트 기반 파일시스템 구조

위의 그림과 같이 일반 파일시스템은 블록 장치를 사용하여 동작한다. 이를 위해 호스트는 게스트가 가상 블록 장치로 사용할 파일을 생성하여 관리한다. 즉, 게스트는 해당 파일을 가상 블록 장치로 인식하여 파일시스템으로 마운트해서 논리적인 구조를 볼 수 있지만, 호스트는 가상 블록 장치를 하나의 독립적인 파일로 인식하기 때문에 게스트가 바라보는 논리적인 구조를 볼 수 없다. 하지만 호스트 기반 파일시스템은 호스트가 게스트에게 제공하는 가상 블록 장치를 사용하는 것이 아니라 호스트의 파일시스템의 일부분을 게스트가 직접 마운트해서 사용하게 된다. 예를 들어, 호스트의 특정 파일시스템에 존재하는 file1과 file2는 호스트 커널에서 inode와 address_space와 같은 자료 구조를 이용하여 관리하는 일반적인 파일이다. 하지만 호스트는 게스트에게 해당 파일들의 inode와 address_space에 대한 접근 권한을 줌으로써 file1과 file2는 게스트에서도 직접 접근이 가능해진다. 이럴 경우, 호스트는 호스트 기반 파일시스템을 사용하는 게스트가 페이지 캐시로 어떤 파일의 어느 위치를 사용하는지를 쉽게 알 수 있기 때문에 Satori에서 했던 것처럼 쉽게 페이지 캐시를 공유할 수 있게 된다. 그리고 게스트 입장에서는 불필요한 가상 블록 장치 접근이 사라졌기 때문에 기존의 호스트와 게스트의 블록 드라이버 계층을 모두 통과해야 하는 접근 중복 문제를 해결할 수 있다.

5. 실험 및 평가

우리는 호스트 기반 파일시스템의 접근 중복성 제거에 따른 성능 향상을 평가하기 위해 리눅스에서 일반적으로 사용하는 유틸리티들을 이용하여 호스트와 게스트의 일반 파일시스템과 비교 및 평가하였다. 자세한 실험 환경은 다음과 같다.

CPU	Intel i7
Cache	8Mbyte
Memory	4Gbyte
OS	Ubuntu 11.04
HFS	Ext4
GFS	Ext4
MyFS	Ext4 (host side)

실험은 호스트 캐시를 사용한 경우와 사용하지 않은 경우로 나누어서 진행하였으며, 실험 대상은 호스트의 일반 파일시스템(HFS), 게스트의 일반 파일시스템(GFS) 그리고 호스트 기반 파일시스템(MyFS) 이다. GFS에서 호스트 캐시는 호스트에서 관리하는 가상 블록 장치 파일이 호스트 내부에 페이지 캐시로 존재하는지를 나타내고, MyFS에서는 게스트가 공유하는 호스트 파일시스템의 파일들이 페이지 캐시에 존재하는지를 나타낸다. 자세한 실험 결과는 다음 그림과 같다.

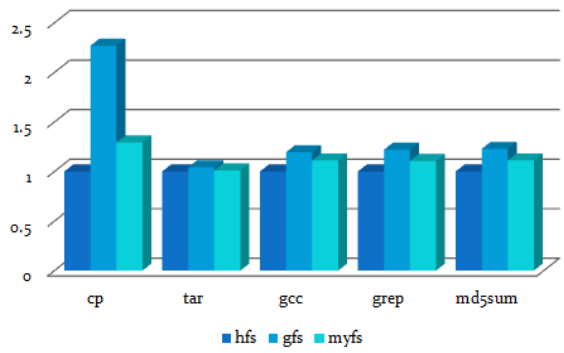

그림 2 벤치마킹 (호스트 캐시 미사용)

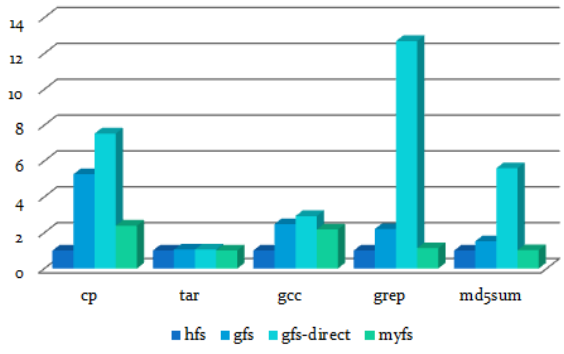

그림 3 벤치마킹 (호스트 캐시 사용)

위의 그림과 같이, 호스트 캐시를 사용하지 않은 경우에는 성능 차이가 적게 나타난다. 이는 호스트에서 실제 블록 장치를 통해서 페이지 캐시를 채우는 작업이 게스트에서 가상 블록 장치를 통해서 페이지 캐시를 채우는 작업에 비해 월등히 크기 때문에 상대적으로 HFS와 MyFS에 비해 성능 차이가 적게 나타나는 것처럼 보인다. 하지만 호스트 캐시를 사용하는 경우에는 GFS와 MyFS가 큰 성능 차이를 보이고 있다. 이는 MyFS는 게스트에서 가상 블록 장치없이 호스트에 존재하는 페이지 캐시를 바로 매핑해서 사용하지만, GFS는 게스트의 가상 블록 장치를 통해서 호스트의 페이지 캐시를 게스트의 페이지 캐시로 다시 읽어들이기 때문에 그 성능 차이가 커지게 된다. 또한, GFS는 MyFS에 비해 메모리 사용량이 두 배가 된다. 왜냐하면 MyFS는 호스트의 페이지 캐시를 공유하지만, GFS는 호스트의 페이지 캐시와는 별도로 페이지 캐시를 따로 관리하기 때문이다. 그래서 메모리 사용량을 동일하게 하기 위해 호스트의 페이지 캐시를 사용하지 않은 GFS-Direct의 성능은 MyFS에 비해 월등히 떨어지게 된다.

이처럼 제안 파일시스템은 논리적인 정보를 이용하여 페이지 캐시를 효과적으로 공유할 수 있을 뿐 아니라 가상 블록 장치 우회를 통해 호스트의 페이지 캐시를 직접 공유함으로써 성능 면에서도 이점을 가지게 된다.

6. 결론

본 논문에서는 가상 환경에서 메모리와 스토리지의 자원 중복성을 제거하기 위해 호스트 기반 파일시스템을 제안하였다. 이는 호스트와 게스트간의 시멘틱 갭을 제거하여 호스트가 게스트의 파일시스템의 논리적인 구조를 공유함으로써 가상 블록 장치를 사용하는 게스트에 비해 자원 공유가 쉽고, 가상 블록 장치에 대한 접근 시간을 제거하여 성능이 좋아진다. 향후에는 해당 연구를 확장하여 메모리 반환 기법, 동적 메모리 분배 및 호스트 스와핑 등에 적용할 계획이다.

참고문헌

[1] S. Nanda, T. Chiueh, and S. Brook "A Survey on Virtualization Technologiecs," RPE Report, http://citeseer.ist.psu.edu/720518.html, 2005.

[2] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, “Xen and the Art of Virtualization,” Proc. Of 9th ACM Symposiumon Operating Systems Principles, 2003.

[3] S. T. Jones, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, “Geiger: Monitoring the buffer cache in a virtual machine environment,” Proc. Of 12th Architectural Support for Programming Languages and Operating Systems, 2006.

[4] C. A. Waldspurger, “Memory resource management in VMware ESX server,” Proc. Of 5th USENIX symposiumon Operating System Design and Implementation, 2002.

[5] G. Miło´s, D. G. Murray, S. Hand, and M. A. Fetterman, “Satori: Enlightened page sharing,” USENIX Annual Technical Conference, 2009.